STAT 378 Assignment 4

Siwei Li Siqi Wang Ruhan Zhang

October 27, 2015

Step1: Main Algorithmm for Beta distribution-Metropolis Hasting

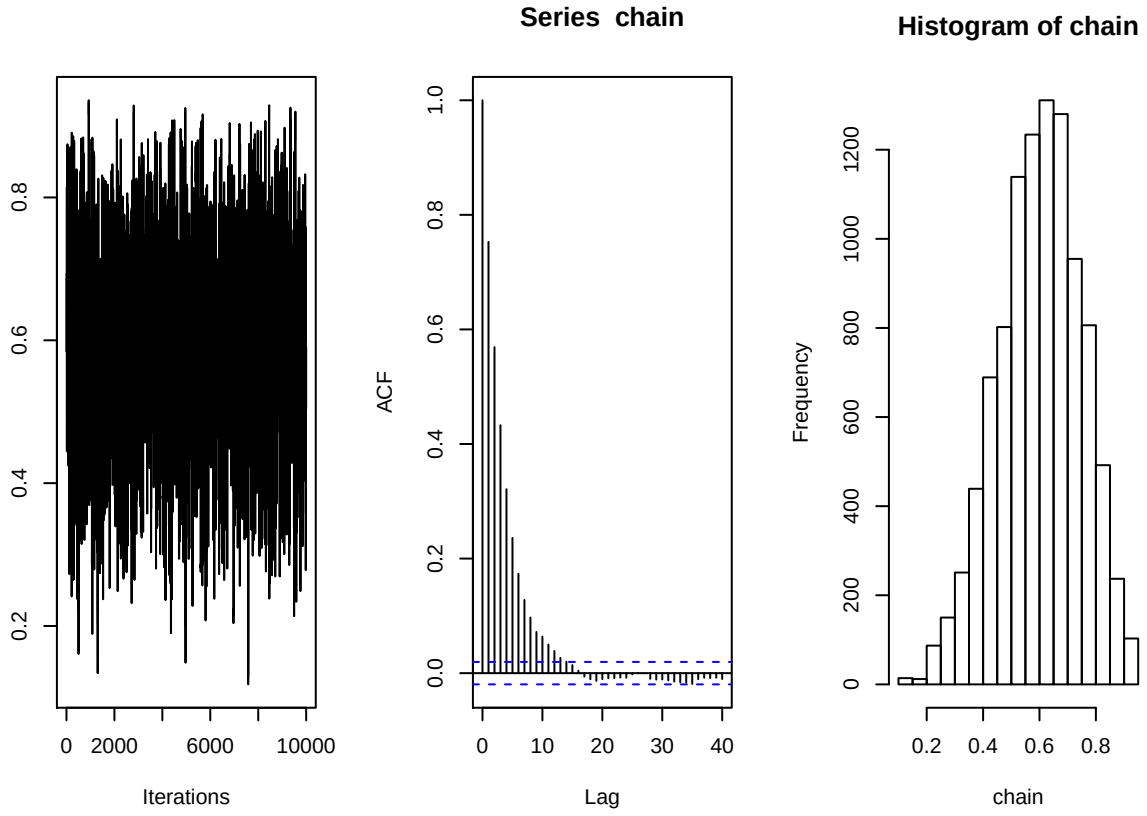
```
set.seed(99)
#prepare for the traceplot in later stages:
library(coda)
```

```
## Warning: package 'coda' was built under R version 3.1.3
```

```
#Metropolis Hasting Procedure:
Beta_Metropolis <- function(alpha=6,beta=4,c,iterations){
  chain <- numeric(iterations+1)
  start_value <- runif(1)
  chain[1] <- start_value
  for(i in 1:iterations){
    current_value <- chain[i]
    #proposal function:
    newbeta <- 1
    while (newbeta == 0 | newbeta == 1) {
      newbeta <- rbeta(1, c*current_value, c*(1-current_value))
    }
    #proposal ratio:
    proposal_ratio <- dbeta(current_value, c*newbeta, c*(1-newbeta)) / dbeta(newbeta, c*current_value, c
    #posterior ratio:
    posterior_ratio <- dbeta(newbeta, alpha, beta) / dbeta(current_value, alpha, beta)
    #acceptance ratio:
    if(runif(1) < min(1, posterior_ratio * proposal_ratio)){
      chain[i+1] <- newbeta
    }else{
      chain[i+1] <- current_value
    }
  }
  return(chain)
}
```

Step2: evaluate the performance of the sampler

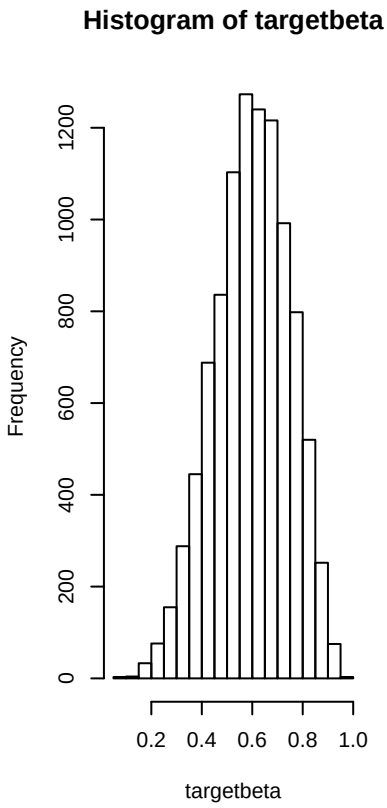
```
chain <- Beta_Metropolis(alpha=6,beta=4,c=1,iterations=10000)
par(mfrow=c(1,3))  #1 row, 3 columns
traceplot(as.mcmc(chain)); acf(chain); hist(chain)  #plot commands
```




```
#graphical comparison
targetbeta <- rbeta(10000,6,4)
hist(targetbeta)
#numerical comparison - Kolmogorov-Smirnov statistic
ks.test(chain,targetbeta)
```

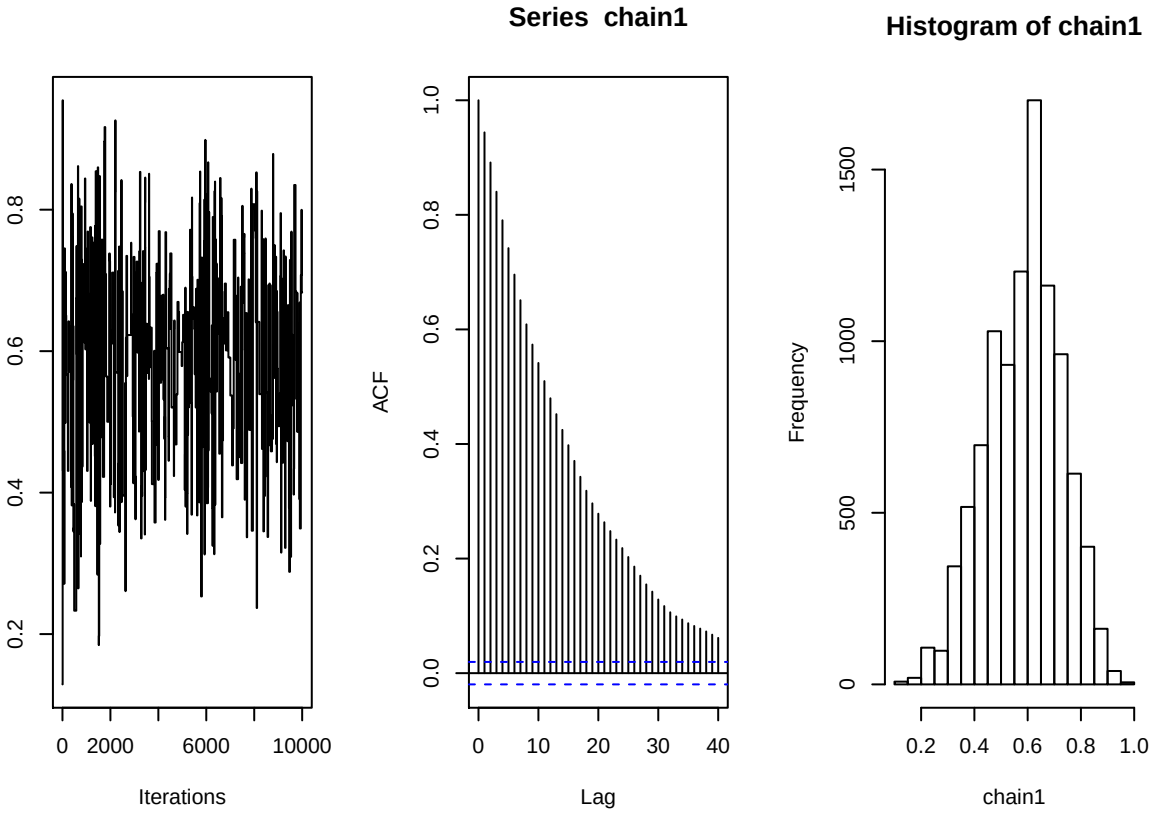
```
## Warning in ks.test(chain, targetbeta): p-value will be approximate in the
## presence of ties
```

```
##
##  Two-sample Kolmogorov-Smirnov test
##
## data:  chain and targetbeta
## D = 0.0193, p-value = 0.04914
## alternative hypothesis: two-sided
```

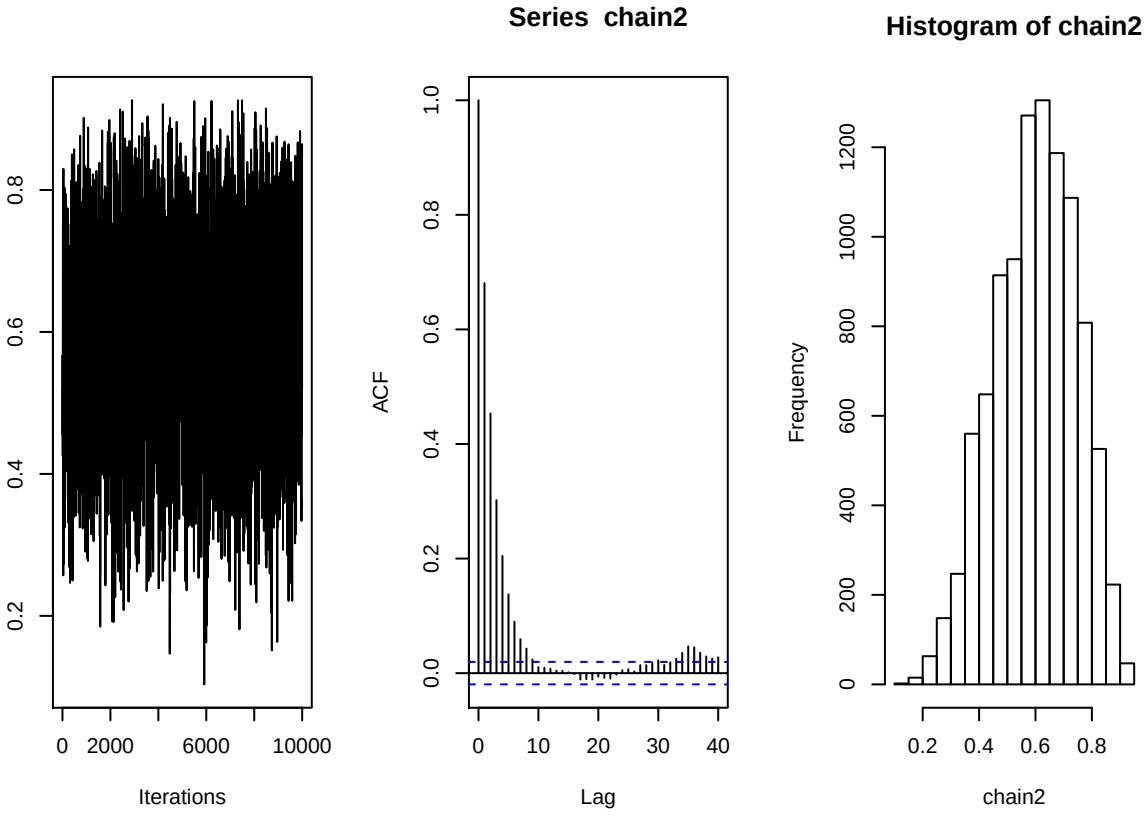


Step3: re-run the sampler with c=0.1, c=2.5 and c=10

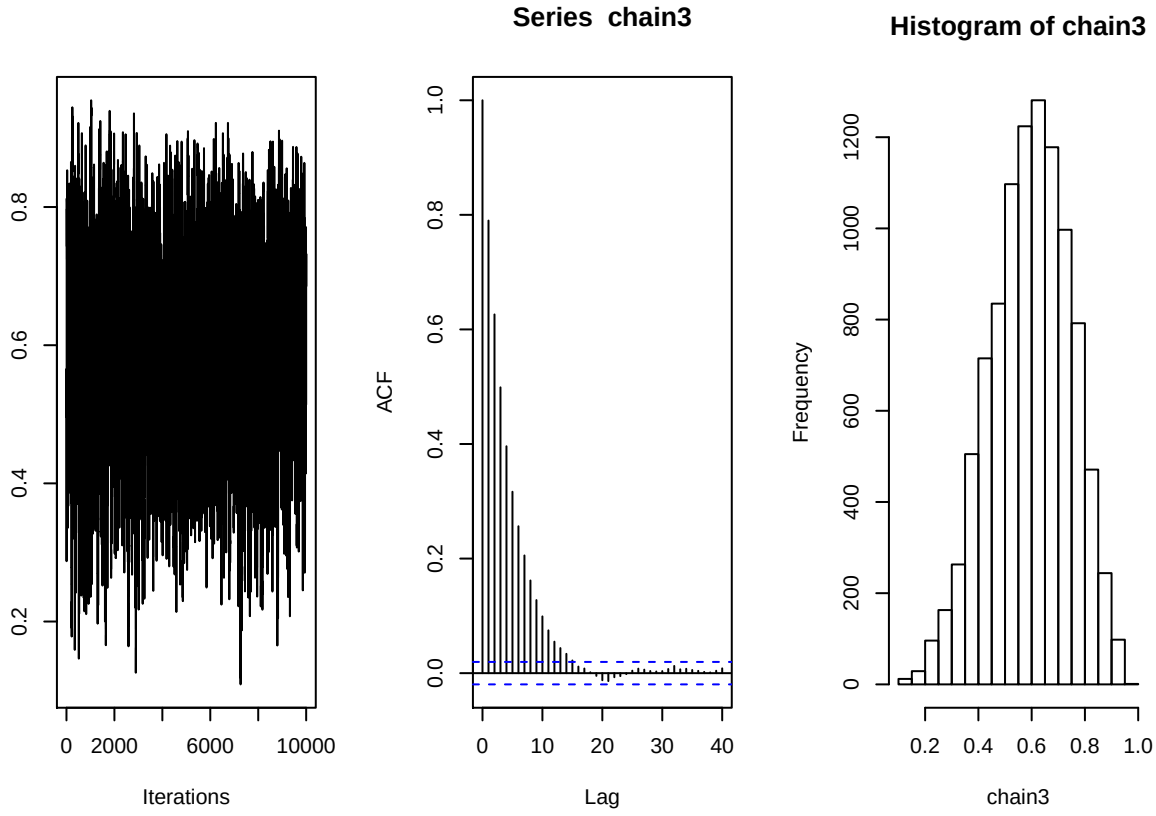
```
par(mfrow=c(1,3))
chain1 <- Beta_Metropolis(alpha=6,beta=4,c=0.1,iterations=10000)
traceplot(as.mcmc(chain1)); acf(chain1); hist(chain1)
```




```
chain2 <- Beta_Metropolis(alpha=6,beta=4,c=2.5,iterations=10000)
traceplot(as.mcmc(chain2)); acf(chain2); hist(chain2)
```




```
chain3 <- Beta_Metropolis(alpha=6,beta=4,c=10,iterations=10000)
traceplot(as.mcmc(chain3)); acf(chain3); hist(chain3)
```



To sample from the full conditioncal distribution, we need to figure out the full condition density function first. The problem indicates that

$$p(x|y) \propto y\mathrm{e}^{-yx} \quad 0 < x < B < \infty$$
$$p(y|x) \propto x\mathrm{e}^{-yx} \quad 0 < y < B < \infty$$

Suppose the normalizing constant of $p(x|y)$ is c, we can figure out c by

$$\int_0^B cy\mathrm{e}^{-yx}dx = 1$$
$$c(-\mathrm{e}^{-yx}|_0^B) = 1$$
$$c(1-\mathrm{e}^{-By}) = 1$$

So now we know that $c = \frac{1}{1-\mathrm{e}^{-By}}$. Similarly, we can find that the normalizing constant for $p(y|x)$ is $\frac{1}{1-\mathrm{e}^{-Bx}}$. And we can write the density function as

$$p(x|y) = \frac{y\mathrm{e}^{-yx}}{1-\mathrm{e}^{-By}} \quad 0 < x < B < \infty$$

$$p(y|x) = \frac{x\mathrm{e}^{-yx}}{1-\mathrm{e}^{-Bx}} \quad 0 < y < B < \infty$$

Then the cumulative density functions are

$$F_{X|Y}(x|y) = \frac{1-\mathrm{e}^{-yx}}{1-\mathrm{e}^{-By}} \quad 0 < x < B$$

$$F_{Y|X}(y|x) = \frac{1 - \mathrm{e}^{-yx}}{1 - \mathrm{e}^{-Bx}} \quad 0 < y < B$$

With some computations, we may obtain the inverse function of two CDFs:

$$F^{(-1)}_{X|Y}(u|x,y) = \frac{-\log\left(1 - u + u\mathrm{e}^{-By}\right)}{y} \quad 0 < x < B$$

$$F^{(-1)}_{Y|X}(u|x,y) = \frac{-\log\left(1 - u + u\mathrm{e}^{-Bx}\right)}{x} \quad 0 < y < B$$

where $0 \leqslant u \leqslant 1$. Now, we can list the algorithm scheme:

1. Select initial values (x_0, y_0) and set the iteration number T.
2. Set index $t = 1$.
3. Generate random draws (x_t, y_t) where
   - Generate two numbers u_1, u_2 from Unif(0,1)
   - Let $x_t = F^{(-1)}_{X|Y}(u_1|x_{t-1}, y_{t-1})$
   - Let $y_t = F^{(-1)}_{Y|X}(u_1|x_t, y_{t-1})$
4. Record sample (x_t, y_t)
5. Set $t = t + 1$, return to step 3 if $t \leqslant T$.
6. Return T samples.

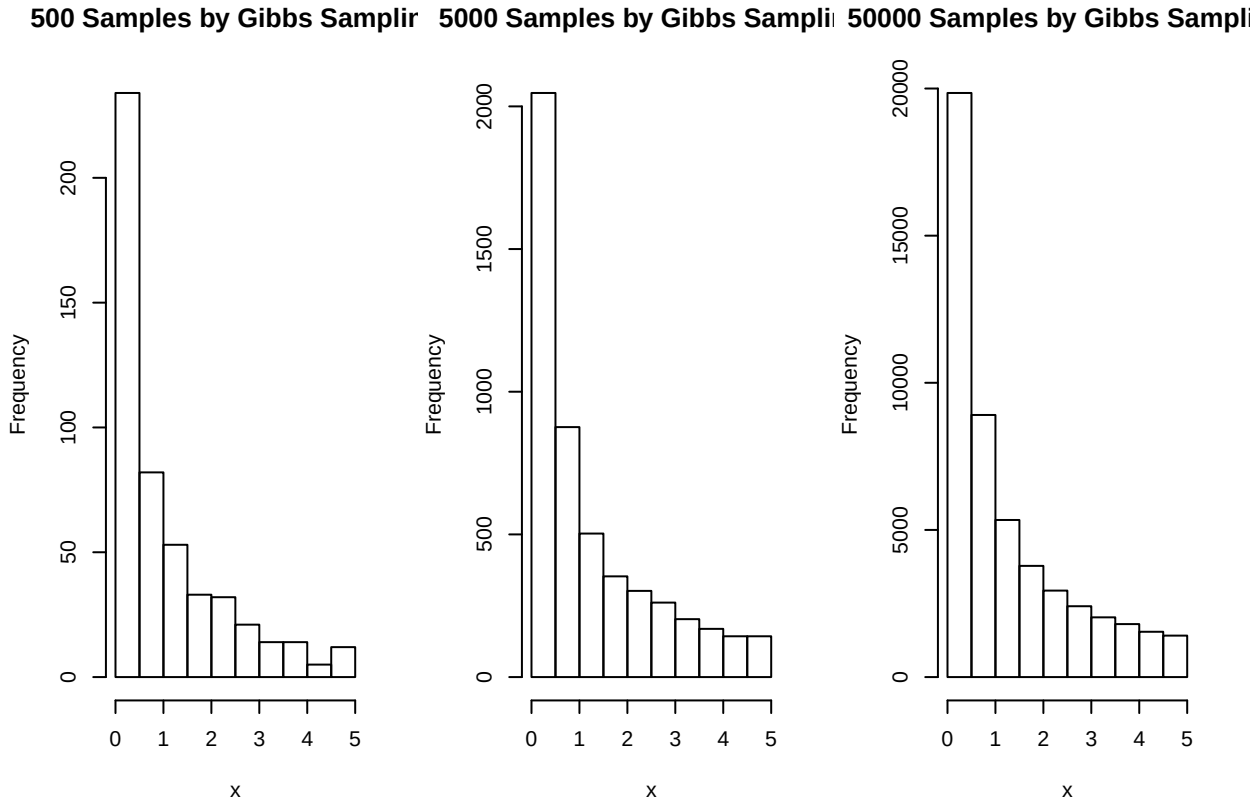
The following is the R code of the algorithm, we denote the Gibbs sampling function as Gibbs.sim():

```
Gibbs.sim <- function(x0,y0,B,Ite){#The inputs are initial values x0, y0, the bound B and the iteration
  if (x0<0 || x0>B ||y0<0 || y0>B ) {#Check the validity of initial values
    stop("The initial value is not in the correct range.")
  }
  else {x_result=c()
        y_result=c()
        xt=x0
        yt=y0
        for(t in 1:Ite){
          u1=runif(1)
          u2=runif(1)
          xt=-log(1-u1*(1-exp(-B*yt)))/yt
          yt=-log(1-u2*(1-exp(-B*xt)))/xt #the xt here is the updated why, thus the algorithm construct
          x_result=c(x_result, xt)
          y_result=c(y_result, yt)
  }
  return(cbind(x_result,y_result)) #We will do burn in later
  }
}
```

We do not let the function automatically do the burn-in since we think it is not proper to discard the sampling information easily. Besides, the burn-in process is not difficult to be implemented in this case. Now for $B = 5$, we made three samples with sizes $T = 500, 5000, 50000$ and plot the histograms. For each sample, we burn-in the first 200 samples. In order to ensure the correct sample amount, we acutually did 700, 5200, 50200 iterations each time.

```r
set.seed(930817)
sample500=Gibbs.sim(1,2,5,700)[-(1:200),] #We burn-in the first 200 samples in order to get 500 samples
sample5000=Gibbs.sim(1,2,5,5200)[-(1:200),] #Also burn-in the first 200 samples
sample50000=Gibbs.sim(1,2,5,50200)[-(1:200),] #50000 samples after burn-in

### Plot the histogram ###
par(mfrow=c(1,3))
hist(sample500[,1],xlab="x", main="500 Samples by Gibbs Sampling")
hist(sample5000[,1],xlab="x", main="5000 Samples by Gibbs Sampling")
hist(sample50000[,1],xlab="x", main="50000 Samples by Gibbs Sampling")
```



We can see that all the histograms are skewed to the right. The shape of each plot is similar. \

By Gibbs sampling, once the Markov Chain contructed by the algorithm converges to the target distribution which we want to find, we can treat samples as approximates from the target distribution. Thus we can use the mean of samples of x to estimate $E_{p(X)}[X]$. Here are the estimates from three sizes samples:

```r
mean(sample500[,1])
```

```
## [1] 1.073697
```

```r
mean(sample5000[,1])
```

```
## [1] 1.250335
```

```r
mean(sample50000[,1])
```

```
## [1] 1.270836
```

The mean of the sample with $T = 500$ is about 1.07 and the other two means are approximately 1.25 and 1.27. We can see that mean estimates of samples with $T = 5000$ and $T = 50000$ are much closer. It is a reasonable phenomena since as the iteration number increases, we will get a more accurate approximation from the target distribution. It indicates that we may get a good estimate of $E_{p(X)}[X]$ from samples with $T > 5000$.